Gayathri Ravichandran

5437937197

Scala Version : 2.11

Spark: 2.3.1

TASK 1:

To run : ./spark-submit --class Gayathri_Ravichandran_SON_Task1 frequentitems_2.11-0.1.jar <path to input file> <support> <output file name>

Support	Time in seconds
30	562
40	61

TASK 2:

To run: ./spark-submit --class Gayathri_Ravichandran_SON_Task2 frequentitems_2.11-0.1.jar <path to input file> <support> <output file name>

Execution Time

Support	Time in seconds
500	33
1000	15

TASK 3

./spark-submit --class Gayathri_Ravichandran_SON_Task3 frequentitems_2.11-0.1.jar <path to input file> <support> <output file name>

Support	Time in seconds
100000	545
120000	486

Algorithm Details:

To find the frequent set of words, I implemented the Apriori Algorithm. After partitioning the dataset using the .mapParitions() function, I send each partition to my apriori function. The apriori function first finds the counts of individual words throughout the partition. I get the candidates by comparing the count to the threshold, where threshold = support/number of partitions. After finding single word candidates, I use the combinations() function to generate all possible combinations of words from these candidate sets of different lengths iteratively. At each step, I compare the count of these combinations and check whether the count >= threshold. If yes, I add them to my candidate set. In this way, I got my

candidate set and then I found the true counts in the entire dataset. Thereafter, I sorted the candidate set lexicographically and then wrote it to a file.

BONUS QUESTION:

When you use a smaller threshold, a large number of candidates are generated , which would slow down the collect() operation on the large size RDDs. Hence, we use a larger threshold so as to limit the number of candidates generated which would speed up the collect() operation.